Cheatography

VSS Cheat Sheet

by dionysius via cheatography.com/34418/cs/10782/

Naming Terminology

Name	Zeichenkette welche auf eine Entity verweist
Address	Name eines Access Points
Access Point	Spezieller Typ einer Entity um auf andere Entity zuzugreifen
Entity	Alles was betrieben werden kann

Address und Access Points können ändern, aber weiterhin erreichbar via Name. Entity kann auf mehrere Access Points hören.

Naming Beispiele

System	Name	Address	Access Point	Entity
Post	Recipient's name	City + ZIP Code + Street + Number	Mailbox	Recipient (Person)
File System	Path	File Handle	Disk Drive	File (String of bytes)
DNS	Fully Qualified Domain Name	IP	Network Interface	Host
ARP	IP	MAC	Network Interface	Host

Flat Naming

Names sind unstrukturierte Identifiers

Meistens:

* identifiziert Entity unique

* kann Random sein

Können zwar eine interne Struktur haben, wird jedoch nicht für Name Resolution verwendet.

Broadcasting

+ Selbstverwaltetes Naming system

+ Einfach zu implementieren

+ Ortsunabhängig

- Skaliert nicht (Overhead Kommunikation entspricht Anzahl Entity)

Recursive Resolution

+ Besseres Caching möglich

- Hohe Performance Ansprüche an jedem Nameserver

Structured Naming

Hierarchisch strukturiert, z.B. unterricht.hsr.ch

Namensauflösung kann auf Layers mit versch. Verantw. und Anf. aufgeteilt werden

Attribute-Based Naming

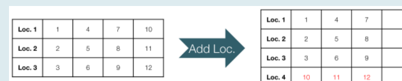
Suche nach Entity aufgrund Attribut-Paare

Distributed Hash Tables

Spezielle Art von Hashing, welches Vergrösserung erlaubt

Im Schnitt nur Anz.Keys/Anz.Nodes zu verschieben

Distributed Hash Tables Beispiel



Vergrösserung von 3 auf 4 Nodes, nur 3 Keys verschieben

P2P Networks

Peer	Participant, Equal, may act as server and/or client
Decentralized	
Overlay network	
Structures	Unstructured, Structured, Hybrid

Torrent Fields

Feld	Wert	Beschreibung
Announce	http://<IP>:80	URL des/der Tracker
Info	<Block von Daten>	Ein Dictionary (Bencoding) für die weiteren Informationen (siehe unten)
Name	vss_<Gruppe>_relaxing_music.mp4	Datei oder Verzeichnisname des Contents
Piece Length	32768	Anzahl Bytes pro Teil (in der Regel 2er-Potenz)
Pieces	<Hashes> (7240 bytes lang)	Eine Liste (Bencoding) der Hash-Werte der Teile
Length	9654370	Gesamtgrösse der Datei (Sofern der Content nur eine Datei ist)
Files	Nicht vorhanden	Ein Dictionary mit Pfaden und Datei-Längen (Sofern der Content mehrere Dateien ist)
Path	Nicht vorhanden	Eine Liste mit Teilen eines Pfades. Teil vom Files-Feld

Torrent Q&A

Welche Aufgabe erfüllt ein Peer?

> Lädt Daten von anderen Peers. stellt fertige Pieces seinem Schwarm zur Verfügung. Auswahl der nächsten Teile verantwortlich. Fertiges Teil sicherstellen indem SHA-1 Hashs überprüfen.

Weshalb sind Files in Teile aufgeteilt?

> Von mehreren gleichzeitig laden. Upload schon nach fertigem Teil. Fehler im Teil verwirft nur Teil.

Auswahl der Teile folgt vier Policies

> **Strict Priority**: Ein Teil wird komplett geladen damit möglichst schnell verifiziert und weitergegeben werden kann. **Rarest First**: Seltene Teile werden bevorzugt um Engpässe zu verhindern. **Random First Piece**: Damit nicht zuerst ein seltenes Teil angefordert wird, würde sonst Upload verzögern und zu Choking führen. **Endgame Mode**: Verbleibende Teile mehrmals parallel von div. Peers angefordert um Download schnell abzuschliessen.

Chord: Storing Resources

succ(_) nicht verwechseln mit "_.successor"

succ(_) ist eine verteilte Funktion

_.successor ist eine lokale Funktion

Für alle p gilt: p.successor = succ(p+1)

Chord: Finger Table

Wenn korrekt, lookup ist O(log N)

Wenn successor pointers (p.finger[1]) korrekt. lookup ergibt korrektes Resultat

Aber gleichzeitige joins oder failing nodes korruptieren finger table

* periodische Stabilisation Prozedur

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Chord: Finger Table (cont)

* dank Stablisation, Netzwerk konvergiert in konsistenten Zustand

Chord: Stablize

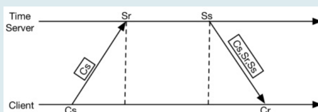
Example: p.successor does not point to its immediate successor (p < x & x < q)

Example: p.predecessor is outdated, p is notified by o (n < o & o < p)

Chord: Replication

- Possible solution: Replicate files r times by generating r keys for each file.
- Store replica i ∈ {0, …, r - 1} at key (hash(fileURL) + i * 2^m/r) % 2^m
- Periodically run background job to verify and make sure that all copies can still be found.

NTP



δ round-trip time = (Cr-Cs)-(Ss-Sr)

θ offset = Ss-Cr+(δ/2) = ((Ss+Sr)-(Cs+Cr))/2

NTP client passt Uhr an durch verlangsamern oder verschnellern des Intervalls

Logical Clocks: Causuality

Causality is defined using the *happens before* relation "→": a → b (event a happens before event b)

a → b iff at least one of the following conditions are true:

- a and b are in the same process and a occurs before b
- a is the event of sending a message and b is the event of receiving the same message
- ∃ x. a → x ∧ x → b (transitive closure)

→ is a *strict partial order* (think: DAG)

- transitive: (a → b ∧ b → c) ⇒ a → c
- irreflexive: a ↛ a
- asymmetric: a → b ⇒ b ↛ a

Definition of the *concurrent* relation "‖": a ‖ b ⇔ (a ↛ b ∧ b ↛ a)

Running Example: replicated DB

All nodes must ACK the message of updates to other nodes. Perform update only when all have ACKd.

All update messages get timestamp of physical clock. Updates in priority queue (based on timestamp)

Totally ordered multicasting. Use Lamport's Clock as the timestamp

Use a vector clock to version updates. concurrent updates can be merged.

Only Solution Abschnitte hier abgebildet.

Bitcoin Protocol

Protocol: **participate**
- ☐ Relay valid transactions.
- ☐ Relay valid blocks in the longest chain.
- ☐ Work with the longest chain.

Protocol: **miners**
- ☐ Collect valid transactions.
- ☐ Publish valid blocks which extend the longest chain.

Distributed Batch Processing

Motivation: Requirements

A model for processing large, distributed data sets that is…

- Responsive: Jobs respond in a timely manner
  - Reduce network round trips
- Resilient: Jobs succeed when individual nodes fail
  - Redistribute workload
  - Recover lost results
- Scalable: Cost effective increases of both input size and number of computing nodes
  - Minimize required modifications of the code when scaling
  - Flexible partitioning

MapReduce Definition

- Data Model: Key/value pairs `(Key, Val)`
- A job consists of 3 steps/functions:
  - **Map:** `(Key1, Val1) => List[(Key2, Val2)]`
  - Shuffle: `List[(Key2, Val2)] => List[(Key2, List[Val2])]`
  - **Reduce:** `(Key2, List[Val2]) => (Key3, Val3)`
- Framework takes care of partitioning and synchronization
- Jobs can be composed to solve more complex problems

Lamport's Logical Clock

- Each process P_i maintains an internal counter C_i(a) that assigns a number to all events a occurring in process i.
- The (distributed) function C(.) assigns the tuple C(a) to any event a, where C(a) = (C_i(a), i) where a is an event in process i.
- The lexicographic ordering is used for these tuples.
- Properties of C(.):
  - Total: (C(a) < C(b)) ∨ (C(a) > C(b)) ∨ (C(a) = C(b)) (i.e. all elements can be compared) (think: List)
  - Preserves causality: a → b ⇒ C(a) < C(b)
- **Result:** With C(.) we have a global (distributed) counter (timestamp) that is total and preserves causality!

Jeder Pi erhöht Ci(.) bei jedem Event

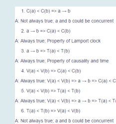
Bei Erhalt: Ci(a) = max(Ci(a), Ci(b))+1

Logical Clocks: Summary

- Causality: A happens before relation between events
- Lamport's Logical Clock: Assigns a timestamp to events that defines a total order amongst them
  - a → b ⇒ C(a) < C(b), but not C(a) < C(b) ⇒ a → b
- Vector Clock: Assigns a timestamp to events that defines a causal order amongst them
  - V(a) < V(b) ⇔ a → b

Erfasst "->" das consept of casuality immer komplett? Nicht immer: Events auf gleichem Prozess können independent sein; Events auf versch. Prozessen können concurrent sein, aber beiläufig durch Externes

Casuality Beispiele



1. C(a) < C(b) => a → b

A: Not always true; a and b could be concurrent

2. a → b => C(a) < C(b)

A: Always true; Property of Lamport clock

3. a → b => T(a) < T(b)

A: Always true; Property of causality and time

4. V(a) < V(b) => C(a) < C(b)

A: Always true; V(a) < V(b) => a → b => C(a) < C(b)

5. V(a) < V(b) => T(a) < T(b)

A: Always true; V(a) < V(b) => a → b => T(a) < T(b)

6. T(a) < T(b) => V(a) < V(b)

A: Not always true; a and b could be concurrent

Does totally ordered multicasting preserve casuality? Yes, Since messages are delivered in the order of their Lamport timestamps, which preserve the happened-before relation.

Resilient Distributed Datasets

Resilient Distributed Datasets

An RDD[A] is a collection of items of type A that is:

- Immutable: Transformations create new RDDs
- Lazy: Operations are executed on demand
- Distributed: Partitioned across a set of nodes
- Cacheable: Intermediate results can be cached to disc/memory
- Resilient: Partition can be rebuilt if data is lost

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

VSS Cheat Sheet

by dionysius via cheatography.com/34418/cs/10782/

Reduce RDD[A]

```
reduce:
(RDD[A], (A, A) => A)) => A
```

- The reduce function must be
  - commutative: `f(a, b) = f(b, a)`
  - associative: `f(f(a, b), c) = f(a, f(b, c))`
- Reduce is easy to parallelize/distribute because order of application does not matter

ints: 1, 2, 3, 4

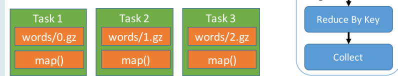
2 + 3 = 5

1 + 5 = 6

6 + 4 = 10 → sum: 10

Tasks

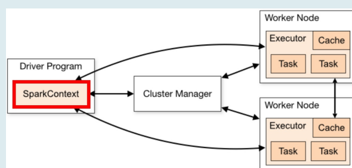
- Data + Computation
- One task per partition and stage
- Scheduled on preferred location node of partition



Stage 1: Hadoop RDD → Map

Stage 2: Reduce By Key → Collect

Deploying on a Cluster



Deploying Naming

Spark Context	Entry point for creating RDDs, Manages connection to the cluster, accesses services like schedulers and block manager
Cluster Manager	Service managing resources in the cluster, e.g. spark standalone, mesos, YARN
Worker Node	A node that can run coe of a Spark application
Executor	A process launched for a certain application, Applications are isolated from each other

MapReduce Summary

- Data parallel mappers and reducers
- Synchronize during shuffling
- A lot of criticism (mostly specific to Hadoop MapReduce)
  - Verbose and imperative API
    - Defining simple jobs already requires pages of code
  - Technical “shortcomings”
    - Disk I/O, data is usually spilled to disk when shuffling
    - Designed for relatively computational intensive mappers

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

https://apollopad.com